



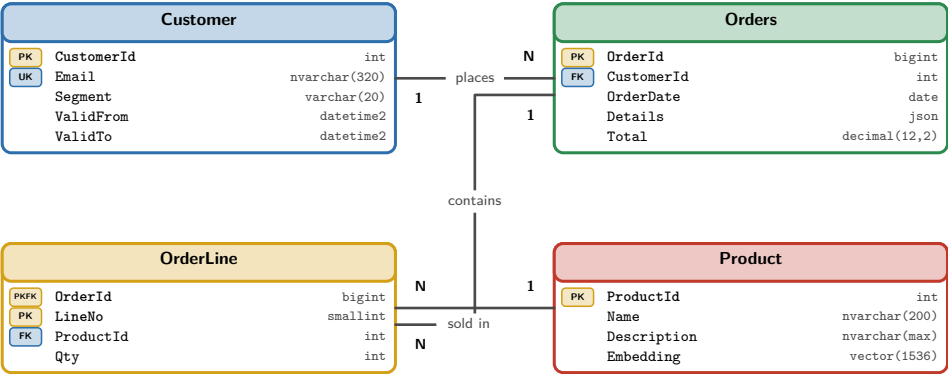
DP-800 Domain 1a: Database Objects and Specialised Tables

DP-800 study guide, 'Design and implement database objects'; SQL Server 2025, Azure SQL and Fabric SQL database docs, as of 2026 | Data types, indexes and columnstore, in-memory, temporal, external, ledger and graph tables, JSON columns and indexes, constraints, sequences, partitioning

DP-800 Domain 1 T-SQL Temporal Ledger JSON

Tables, types and indexes

Order schema used through the chapter



Customer is temporal (ValidFrom/ValidTo), Orders.OrderId comes from a SEQUENCE, Orders.Details is native json, Product.Embedding is a vector for sheet 06

Data	Type	Size / note
Keys	int (4 B), bigint (8 B)	int ends at 2,147,483,647; uniqueidentifier (16 B) as clustered key fragments unless sequential
Money, quantities	decimal(p,s)	exact; float only for scientific values
Text	nvarchar(n) (2 B per char, UTF-16) or varchar(n) with a UTF-8 collation	(max) up to 2 GB, stored off-row when large
Date and time	date, datetime2(p), datetimeoffset	datetime2(3) = 7 B; avoid legacy datetime
Documents	json	native binary JSON; nvarchar(max) + ISJSON check on older engines
Embeddings	vector(n)	float32 by default, 4 B per dimension (sheet 06)
Row limit	8,060 B in-row	variable-length overflow goes off-row; max 1,024 columns

Index	Use	Watch out
Clustered rowstore	OLTP tables, range scans on the key	one per table; wide keys bloat every nonclustered index
Nonclustered + INCLUDE	seek on filters, cover selected columns	too many slow down writes
Filtered	WHERE Status = 'Open': small index on a hot subset	query predicate must match
Clustered columnstore (CCI)	fact tables, analytics, 10x compression typical	singleton updates are expensive; load in batches ≥ 102,400 rows to compress directly
Nonclustered columnstore	real-time analytics on an OLTP table	extra write cost
Ordered CCI	segment elimination on a date key	order maintained only at build/rebuild

Specialised tables

	In-memory memory-optimized	Temporal	External	Ledger	Graph
Use it for	latch-free hot OLTP, session state, staging	audit history, point-in-time queries, SCD	query Parquet/CSV/Delta in storage without loading	tamper evidence for regulated records	many-to-many relationships, paths
Key T-SQL	WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)	PERIOD FOR SYSTEM_TIME + SYSTEM_VERSIONING = ON	CREATE EXTERNAL DATA SOURCE / FILE FORMAT / TABLE; OPENROWSET; CETAS	WITH (LEDGER = ON) or LEDGER = ON (APPEND_ONLY = ON)	AS NODE / AS EDGE; MATCH; SHORTEST_PATH
Platforms / limits	SQL Server (memory-optimized filegroup), Azure SQL Premium / Business Critical	all; history retention policy optional	SQL Server 2022+, Azure SQL MI; check Azure SQL Database status	cannot be turned off; digests to immutable storage	all; edge constraints CONNECTION (A TO B)

Temporal and ledger tables

```
CREATE TABLE dbo.Customer (
  CustomerId int NOT NULL PRIMARY KEY,
  Email nvarchar(320) NOT NULL UNIQUE,
  Segment varchar(20) NOT NULL DEFAULT 'retail',
  ValidFrom datetime2 GENERATED ALWAYS AS ROW START,
  ValidTo datetime2 GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo))
WITH (SYSTEM_VERSIONING = ON (
  HISTORY_TABLE = dbo.CustomerHistory,
  HISTORY_RETENTION_PERIOD = 2 YEARS));

SELECT * FROM dbo.Customer
  FOR SYSTEM_TIME AS OF '2026-01-01' WHERE CustomerId = 42;
```

```
CREATE TABLE dbo.PaymentAudit (
  PaymentId bigint, Amount decimal(12,2), At datetime2)
WITH (LEDGER = ON (APPEND_ONLY = ON)); -- inserts only
EXEC sys.sp_verify_database_ledger_from_digest_storage;
```

In-memory, external and graph tables

```
CREATE TABLE dbo.Cart (
  SessionId uniqueidentifier NOT NULL
  PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 1048576),
  Payload nvarchar(4000))
WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_ONLY);

CREATE EXTERNAL DATA SOURCE lake WITH (
  LOCATION = 'abs://sales@stlake.blob.core.windows.net',
  CREDENTIAL = LakeMI); -- database scoped credential
SELECT TOP 10 * FROM OPENROWSET(BULK '2026/10/*.parquet',
  DATA_SOURCE = 'lake', FORMAT = 'parquet') AS r;

CREATE TABLE dbo.Person (Id int PRIMARY KEY, Name nvarchar(100)) AS NODE;
CREATE TABLE dbo.Knows (Since date,
  CONSTRAINT ec_knows CONNECTION (dbo.Person TO dbo.Person)) AS EDGE;
```

EXAM TIP

'Prove rows were not altered, even by a DBA' → **ledger** (append-only for logs). 'Query the value as it was on a past date' → **temporal**. 'Extreme insert rate, latch contention on a hot table' → **memory-optimized**. 'Query Parquet in the lake without loading' → **external table / OPENROWSET**. 'Friends of friends, shortest path' → **graph**.

JSON, constraints, sequences

Orders.Details (json column) and the paths worth indexing

```
{
  "channel": "web",
  "customer": {
    "id": 42,
    "tier": "gold"
  },
  "items": [
    {
      "sku": "A-100",
      "qty": 2,
      "price": 19.9
    },
    {
      "sku": "B-220",
      "qty": 1,
      "price": 54.0
    }
  ],
  "tags": [
    "promo",
    "gift"
  ]
}
```

filtered often: JSON index path \$.customer.id

array: OPENJSON(Details, '\$.items') WITH (...)

JSON_CONTAINS(Details, 'promo', '\$.tags') = 1

JSON column, JSON index, constraints and a sequence

```
CREATE SEQUENCE dbo.OrderNo AS bigint START WITH 100000
INCREMENT BY 1 CACHE 50;
CREATE TABLE dbo.Orders (
  OrderId bigint NOT NULL
  CONSTRAINT DF_Orders_Id DEFAULT (NEXT VALUE FOR dbo.OrderNo)
  CONSTRAINT PK_Orders PRIMARY KEY CLUSTERED,
  CustomerId int NOT NULL
  CONSTRAINT FK_Orders_Customer REFERENCES dbo.Customer(CustomerId),
  OrderDate date NOT NULL DEFAULT (CAST(SYSUTCDATETIME() AS date)),
  Details json NOT NULL,
  Total decimal(12,2) NOT NULL CONSTRAINT CK_Orders_Total CHECK (Total >= 0));
CREATE JSON INDEX IX_Orders_Details ON dbo.Orders (Details)
  FOR ('$.customer.id', '$.channel');
-- older engines: computed column + regular index
-- ALTER TABLE dbo.Orders ADD CustId AS
--   CAST(JSON_VALUE(Details, '$.customer.id') AS int);
```

Constraint	Enforces	Remember
PRIMARY KEY	unique, not null; clustered by default	one per table
FOREIGN KEY	value exists in the parent	ON DELETE CASCADE / SET NULL / NO ACTION; index the FK column yourself
UNIQUE	no duplicates	allows one NULL (filtered unique index for 'many NULLs')
CHECK	row-level predicate	WITH NOCHECK leaves it untrusted: the optimiser ignores it
DEFAULT	value when omitted	can call NEXT VALUE FOR a sequence

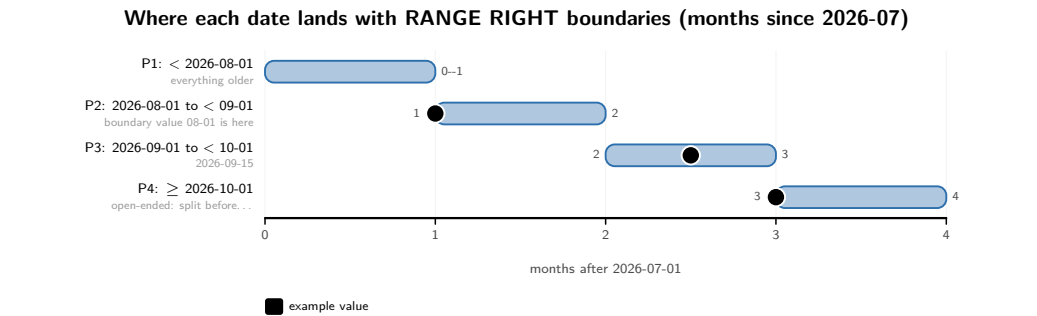
SEQUENCE	IDENTITY
schema object, shared by several tables	column property, one table
value known before the insert (NEXT VALUE FOR)	value known after the insert (SCOPE_IDENTITY(), OUTPUT)
sp_sequence_get_range reserves a block	IDENTITY_INSERT to force values
CACHE 50: fewer disk writes, gaps after a crash	gaps also possible (rollbacks, restarts)

Partition tables and indexes

Monthly RANGE RIGHT partitioning

```

CREATE PARTITION FUNCTION pfMonth (date)
AS RANGE RIGHT FOR VALUES ('2026-08-01', '2026-09-01', '2026-10-01');
CREATE PARTITION SCHEME psMonth
AS PARTITION pfMonth ALL TO ([PRIMARY]); -- Azure SQL DB: PRIMARY only
CREATE TABLE dbo.Sales (SaleDate date NOT NULL, Amount decimal(12,2))
ON psMonth (SaleDate);
SELECT $PARTITION.pfMonth('2026-09-15'); -- returns 3
ALTER TABLE dbo.Sales SWITCH PARTITION 1 TO dbo.Sales_Out;
TRUNCATE TABLE dbo.Sales WITH (PARTITIONS (2));
ALTER PARTITION FUNCTION pfMonth() MERGE RANGE ('2026-08-01');
ALTER PARTITION SCHEME psMonth NEXT USED [PRIMARY];
ALTER PARTITION FUNCTION pfMonth() SPLIT RANGE ('2026-11-01');
```



3 boundaries → 4 partitions. RANGE RIGHT puts each boundary in the partition to its right (natural for dates: the 1st of the month starts it); RANGE LEFT would put 2026-08-01 in P1

Worked example: sliding window

PROBLEM
Keep 24 months of sales, about 40 million rows per month, in a partitioned table. How many partitions, and why switch instead of DELETE for the monthly purge?

1. Rows in the table
⇒ $M \times r = 24 \times 40,000,000 = 960,000,000$
why: steady state

2. Partitions
⇒ 24 data + 2 empty ends = 26 partitions → 25 boundaries
why: empty ends make SPLIT and MERGE metadata-only

3. Purge with DELETE
⇒ 40,000,000 logged row deletes, locks, log growth
why: row by row

3. Purge with SWITCH
⇒ SWITCH oldest → staging, TRUNCATE staging, MERGE RANGE: metadata only, seconds
why: aligned indexes required

GIVEN

- M = 24 months kept
- r = 40,000,000 rows per month
- one empty partition kept at each end

ANSWER
26 partitions (25 boundaries); SWITCH + TRUNCATE + MERGE purges a month in seconds instead of 40 million logged deletes
Check: sys.partitions shows rows = 0 for the switched partition

Traps and recap

MYTH OR FACT?
“Temporal history is purged automatically after a while”

MYTH: History is kept forever unless you set HISTORY_RETENTION_PERIOD
Set a retention (or partition the history table) or it grows without limit.

MYTH OR FACT?
“A JSON column needs nvarchar(max) and ISJSON”

MYTH: The native json type validates and stores binary JSON
On Azure SQL and SQL Server 2025; older engines use nvarchar + ISJSON.

MYTH OR FACT?
“Partitioning always speeds up queries”

MYTH: It mainly speeds maintenance
Queries gain only with partition elimination on the partitioning column.

MYTH OR FACT?
“You can switch ledger off later”

MYTH: Ledger tables cannot be converted back
Decide at creation; dropped ledger tables are kept as dropped ledger tables.

MYTH OR FACT?
“UNIQUE forbids NULLs”

MYTH: It allows exactly one NULL
Use a filtered unique index WHERE col IS NOT NULL for many NULLs.

Q1 Query a customer row as of 1 January 2026?
A SELECT ... FROM dbo.Customer FOR SYSTEM_TIME AS OF '2026-01-01'.

Q2 Insert-only tamper-evident audit log?
A Ledger table WITH (LEDGER = ON (APPEND_ONLY = ON)).

Q3 Durability option for a memory-optimized staging table whose data may be lost on restart?

A DURABILITY = SCHEMA_ONLY.

Q4 Edge table that only links Person to Person?

A AS EDGE with CONSTRAINT ... CONNECTION (dbo.Person TO dbo.Person).

Q5 Speed up filters on a JSON path on SQL Server 2025?

A CREATE JSON INDEX ... FOR ('\$.path') (or a computed column + index on older engines).

Q6 Get an order number before inserting the row?

A NEXT VALUE FOR dbo.OrderNo from a SEQUENCE.

Q7 Partition number of a value?

A \$PARTITION.pfMonth(value).

Q8 Best index for a 500-million-row fact table queried by aggregates?

A Clustered columnstore index.

Q9 Purge a month instantly from a partitioned table?

A SWITCH the partition to a staging table, TRUNCATE it (or TRUNCATE ... WITH (PARTITIONS (n))).

Type by data

decimal money, datetime2, json, vector

Columnstore for analytics

rowstore + nonclustered for OLTP

Pick the special table by the verb

prove: ledger; as of: temporal; hot: in-memory; lake: external; paths: graph

Constraints must be trusted

WITH CHECK, index FKs

Partition for maintenance

RANGE RIGHT for dates, switch to purge